FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

RDFS Rule-based Reasoning

Sebastian Rudolph
RDFS Rule-based Reasoning

User Interface & Applications

Trust

Proof

Unifying Logic

Query: SPARQL

Ontology: OWL

Rule: RIF

RDFS

Data interchange: RDF

XML

URI/IRI

Crypto
RDFS Rule-based Reasoning

- User Interface & Applications
- Trust
- Proof
- Unifying Logic
- Query: SPARQL
- Ontology: OWL
- Rule: RIF
- RDFS
- Data interchange: RDF
- XML
- URI/IRI

TU Dresden

Foundations of Semantic Web Technologies
Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation
Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation
Constituents of Rules

- basic elements of rules are atoms
  - ground atoms without free variables
  - non-ground atoms with free variables
What are Rules?

1. logic rules (fragments of predicate logic):
   - $F \rightarrow G$ equivalent to $\neg F \lor G$
   - logical extension of knowledge base $\rightsquigarrow$ static
   - open world
   - declarative (describing)
What are Rules?

1. Logic rules (fragments of predicate logic):
   - \( F \rightarrow G \) equivalent to \( \neg F \lor G \)
   - logical extension of knowledge base \( \leadsto \) static
   - open world
   - declarative (describing)

2. Procedural rules (e.g. production rules):
   - “If X then Y else Z”
   - executable commands \( \leadsto \) dynamic
   - operational (meaning = effect caused when executed)
What are Rules?

1. logic rules (fragments of predicate logic):
   - $F \rightarrow G$ equivalent to $\neg F \vee G$
   - logical extension of knowledge base $\leadsto$ static
   - open world
   - declarative (describing)

2. procedural rules (e.g. production rules):
   - “If X then Y else Z”
   - executable commands $\leadsto$ dynamic
   - operational (meaning = effect caused when executed)

3. logic programming et al. (e.g. PROLOG, F-Logic):
   - `man(X) :- person(X) AND NOT woman(X)`
   - approximation of logical semantics with operational aspects, built-ins are possible
   - often closed-world
   - semi-declarative
Predicate Logic as a Rule Language

- rules as implication formulae in predicate logic:

\[
H \leftarrow A_1 \land A_2 \land \ldots \land A_n
\]

head \hspace{1cm} body

- implications often written from right to left (← or :-)
- constants, variables and function symbols allowed
- quantifiers for variables are often omitted: free variables are often understood as universally quantified (i.e. rule is valid for all variable assignments)
Predicate Logic as a Rule Language

- rules as implication formulae in predicate logic:

\[ H \leftarrow A_1 \land A_2 \land \ldots \land A_n \]

\( H \) is the head, \( A_1, A_2, \ldots, A_n \) are the body.

\( \leadsto \) semantically equivalent to disjunction:

\[ H \lor \neg A_1 \lor \neg A_2 \lor \ldots \lor \neg A_n \]

- implications often written from right to left (\( \leftarrow \) or \( :\leftarrow \))
- constants, variables and function symbols allowed
- quantifiers for variables are often omitted:
  free variables are often understood as universally quantified
  (i.e. rule is valid for all variable assignments)
Rules – Example

Example:

\[
\text{hasUncle}(x, z) \leftarrow \text{hasParent}(x, y) \land \text{hasBrother}(y, z)
\]

- we use short names (hasUncle) instead of IRIs like http://example.org/Example#hasUncle
- we use \(x, y, z\) for variables
Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation
Lloyd-Topor Transformation

- multiple heads in atoms are usually understood as conjunction

\[ H_1, H_2, \ldots, H_m \leftarrow A_1, A_2, \ldots, A_n \]

equivalent to

\[ H_1 \leftarrow A_1, A_2, \ldots, A_n \]
\[ H_2 \leftarrow A_1, A_2, \ldots, A_n \]
\[ \ldots \]
\[ H_m \leftarrow A_1, A_2, \ldots, A_n \]

- such a rewriting is also referred to as **Lloyd-Topor transformation**
Disjunctive Rules

- some rule formalisms allow for disjunction

\[ H_1, H_2, \ldots, H_m \leftarrow A_1, A_2, \ldots, A_n \]

equivalent to

\[ H_1 \lor H_2 \lor \ldots \lor H_m \leftarrow A_1 \land A_2 \land \ldots \land A_n \]

equivalent to

\[ H_1 \lor H_2 \lor \ldots \lor H_m \lor \neg A_1 \lor \neg A_2 \lor \ldots \lor \neg A_n \]

\( \not\rightarrow \) (not considered here)
Kinds of Rules

names for “rules” in predicate logic:

- **clause**: disjunction of atomic and negated atomic propositions
  - Woman\((x)\) ∨ Man\((x)\) ← Person\((x)\)

- **Horn clause**: clause with at most one non-negated atom
  - Person\((x)\) ← Man\((x)\) ∧ Woman\((x)\)

- **definite clause**: Horn clause with exactly one non-negated atom
  - Father\((x)\) ← Man\((x)\) ∧ hasChild\((x, y)\)

- **fact**: clause containing just one non-negated atom
  - Woman\((x)\)
Kinds of Rules

names for “rules” in predicate logic:

- **clause**: disjunction of atomic and negated atomic propositions
  
  – \( \text{Woman}(x) \lor \text{Man}(x) \leftarrow \text{Person}(x) \)

- **Horn clause**: clause with at most one non-negated atom
  
  – \( \leftarrow \text{Man}(x) \land \text{Woman}(x) \)

  \( \leadsto \) “integrity constraints”
Kinds of Rules

names for “rules” in predicate logic:

- **Clause**: disjunction of atomic and negated atomic propositions
  
  - \( \text{Woman}(x) \lor \text{Man}(x) \leftarrow \text{Person}(x) \)

- **Horn clause**: clause with at most one non-negated atom
  
  - \( \leftarrow \text{Man}(x) \land \text{Woman}(x) \)
  
  \( \rightsquigarrow \) “integrity constraints”

- **Definite clause**: Horn clause with exactly one non-negated atom
  
  - \( \text{Father}(x) \leftarrow \text{Man}(x) \land \text{hasChild}(x, y) \)
Kinds of Rules

names for “rules” in predicate logic:

- **clause:** disjunction of atomic and negated atomic propositions
  - \( \text{Woman}(x) \lor \text{Man}(x) \leftarrow \text{Person}(x) \)

- **Horn clause:** clause with at most one non-negated atom
  - \( \leftarrow \text{Man}(x) \land \text{Woman}(x) \)
  - “integrity constraints”

- **definite clause:** Horn clause with exactly one non-negated atom
  - \( \text{Father}(x) \leftarrow \text{Man}(x) \land \text{hasChild}(x, y) \)

- **fact:** clause containing just one non-negated atom
  - \( \text{Woman}(\text{gisela}) \)
Kinds of Rules

Rules may also contain function symbols:

\[
\begin{align*}
\text{hasUncle}(x, y) & \leftarrow \text{hasBrother}(\text{mother}(x), y) \\
\text{hasFather}(x, \text{father}(x)) & \leftarrow \text{Person}(x)
\end{align*}
\]

⇝ new elements are dynamically generated
⇝ not considered here
⇝ see logic programming
Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation
Datalog

Horn rules without function symbols $\leadsto$ Datalog rules

- logical rule language, originally basis of deductive databases
- knowledge bases ("programs") consisting of Horn clauses without function symbols
- decidable
- efficient for big datasets, combined complexity ExpTime
- a lot of research done in the 1980s
Datalog as Extension of the Relation Calculus

Datalog can be conceived as Extension of the relation calculus by recursion

\[
\begin{align*}
  T(x, y) & \leftarrow E(x, y) \\
  T(x, y) & \leftarrow E(x, z) \land T(z, y)
\end{align*}
\]

\( \rightsquigarrow \) computes the transitive closure (T) of the binary relation E, (e.g. if E contains the edges of a graph)

- a set of (ground) facts is also called an instance
Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation
Semantics of Datalog

three different but equivalent ways to define the semantics:

- model-theoretically
- proof-theoretically
- via fixpoints
Model-theoretic Semantics of Datalog

rules are seen as logical sentences:

\[ \forall x, y. (T(x, y) \leftarrow E(x, y)) \]
\[ \forall x, y. (T(x, y) \leftarrow E(x, z) \land T(z, y)) \]

- not sufficient to uniquely determine a solution
  \[ \Rightarrow \] interpretation of \( T \) has to be minimal
Model-theoretic Semantics of Datalog

in principle, a Datalog rule

$$\rho: R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$$

represents the FOL sentence

$$\forall x_1, \ldots, x_n. (R_1(u_1) \leftarrow R_2(u_2) \wedge \ldots \wedge R_n(u_n))$$

- $x_1, \ldots, x_n$ are the rule’s variables and $\leftarrow$ is logical implication
- an instance $I$ satisfies $\rho$, written $I \models \rho$, if and only if for every instantiation

$$R_1(\nu(u_1)) \leftarrow R_2(\nu(u_2)), \ldots, R_n(\nu(u_n))$$

we find $R_1(\nu(u_1))$ satisfied whenever $R_2(\nu(u_2)), \ldots, R_n(\nu(u_n))$ are satisfied
Model-theoretic Semantics of Datalog

- An instance I is a model of a Datalog program P, if I satisfies every rule in P (seen as a FOL formula).
- The semantics of P for the input I is the minimal model that contains I (if it exists).
- Question: does such a model always exist?
- If so, how can we construct it?
Proof-theoretic Semantics of Datalog

based on proofs for facts:

\[
given: \quad E(a, b), E(b, c), E(c, d) \\
T(x, y) \leftarrow E(x, y) \quad (1) \\
T(x, y) \leftarrow E(x, z) \land T(z, y) \quad (2)
\]

(a) \(E(c, d)\) is a given fact
(b) \(T(c, d)\) follows from (1) and (a)
(c) \(E(b, c)\) is a given fact
(d) \(T(b, d)\) follows from (c), (b) and (2)
(e) \(\ldots\)
Proof-theoretic Semantics of Datalog

- programs can be seen as “factories” that produce all provable facts (deriving new facts from known ones in a bottom-up way by applying rules)
- alternative: top-down evaluation; starting from a to-be-proven fact, one looks for lemmata needed for the proof (⇝ Resolution)
Proof-theoretic Semantics of Datalog

a fact is provable, if it has a proof, represented by a proof-tree:

**Definition**

A proof tree for a fact $A$ for an instance $I$ and a Datalog program $P$ is a labeled
tree in which

1. every node is labeled with a fact
2. every leaf is labeled with a fact from $I$
3. the root is labeled with $A$
4. for each internal leaf there exists an instantiation $A_1 \leftarrow A_2, \ldots, A_n$ of a rule
   in $P$, such that the node is labeled with $A_1$ and its children with $A_2, \ldots, A_n$
Proof-theoretic Semantics of Datalog

based on proofs for facts:

\[ \begin{align*}
\text{given} & : E(a, b), E(b, c), E(c, d) \\
T(x, y) & \leftarrow E(x, y) \\
T(x, y) & \leftarrow E(x, z) \land T(z, y)
\end{align*} \]

(a) \( E(c, d) \) is a given fact
(b) \( T(c, d) \) follows from (1) and (a)
(c) \( E(b, c) \) is a given fact
(d) \( T(b, d) \) follows from (c), (b) and (2)
(e) ...
Fixpoint Semantics

defines the semantics of a Datalog program as the solution of a fixpoint equation

- procedural definition (iteration until fixpoint reached)
- given an instance \( I \) and a Datalog program \( P \), we call a fact \( A \) a direct consequence for \( P \) and \( I \), if
  1. \( A \) is contained in \( I \) or
  2. \( A \leftarrow A_1, \ldots, A_n \) is the instance of a rule from \( P \), such that \( A_1, \ldots, A_n \in I \)
- then we can define a “direct consequence”-operator that computes, starting from an instance, all direct consequences
- similar to the bottom-up proof-theoretic semantics, but shorter proofs are always generated earlier than longer ones
Semantics of Rules

- compatible with other approaches that are based on FOL (e.g. description logics)
- conjunctions in rule heads and disjunction in bodies unnecessary
- other (non-monotonic) semantics definitions possible
  - well-founded semantics
  - stable model semantics
  - answer set semantics
- for Horn rules, these definitions do not differ
- production rules/procedural rules conceive the consequence of a rule as an action “If-then do”
  ~ not considered here
Extensional and Intensional Predicates

• from the database perspective (and opposed to logic programming) one distinguishes facts and rules
• within rules, we distinguish extensional and intensional predicates
• extensional predicates (also: extensional database – edb) are those not occurring in rule heads (in our example: relation E)
• intensional predicates (also: intensional database – idb) are those occurring in at least one head of a rule (in our example: relation T)
• semantics of a datalog program can be understood as a mapping of given instances over edb predicates to instances of idb predicates
Datalog in Practice

Datalog in Practice:
- several implementations available
- some adaptations for Semantic Web: XSD types, URIs (e.g. → IRIS)

Extensions of Datalog:
- disjunctive Datalog allows for disjunctions in rule heads
- non-monotonic negation (no FOL semantics)
Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation
Evaluating Datalog Programs

- top-down or bottom-up evaluation
- direct evaluation versus compilation into an efficient program
- here:
  1. Naïve bottom-up Evaluation
  2. Semi-naïve bottom-up Evaluation
Reverse-Same-Generation

given Datalog program:

\[
\text{rsg}(x, y) \leftarrow \text{flat}(x, y) \\
\text{rsg}(x, y) \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)
\]

given data:

<table>
<thead>
<tr>
<th>up</th>
<th>flat</th>
<th>down</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>g</td>
<td>l</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>a</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>f</td>
<td>n</td>
<td>f</td>
</tr>
<tr>
<td>m</td>
<td>o</td>
<td>g</td>
</tr>
<tr>
<td>g</td>
<td>p</td>
<td>b</td>
</tr>
<tr>
<td>n</td>
<td>m</td>
<td>h</td>
</tr>
<tr>
<td>h</td>
<td>p</td>
<td>c</td>
</tr>
<tr>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>o</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>j</td>
<td>k</td>
<td>k</td>
</tr>
</tbody>
</table>
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Reverse-Same-Generation – Visualization

\[
\text{rsg}(x, y) \leftarrow \text{flat}(x, y)
\]
\[
\text{rsg}(x, y) \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)
\]
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Reverse-Same-Generation – Visualization

\[
\begin{align*}
\text{rsg}(x, y) & \leftarrow \text{flat}(x, y) \\
\text{rsg}(x, y) & \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)
\end{align*}
\]
Reverse-Same-Generation – Visualization

\[ \text{rsg}(x, y) \leftarrow \text{flat}(x, y) \]
\[ \text{rsg}(x, y) \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y) \]
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Reverse-Same-Generation – Visualization

\[
\text{\textit{rsg}}(x, y) \leftarrow \text{\textit{flat}}(x, y)
\]

\[
\text{\textit{rsg}}(x, y) \leftarrow \text{\textit{up}}(x, x_1), \text{\textit{rsg}}(y_1, x_1), \text{\textit{down}}(y_1, y)
\]
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Reverse-Same-Generation – Visualization

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]
Agenda

- Rules
  - Lloyd-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation
Naïve Algorithm for Computing rsg

\[ rsg(x, y) \leftarrow flat(x, y) \]
\[ rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y) \]

**Algorithm 1 RSG**

\[
\begin{align*}
\text{rs}g & := \emptyset \\
\text{repeat} & \\
\text{rs}g & := \text{rs}g \cup flat \cup \pi_{16}(\sigma_{2=4}(\sigma_{3=5}(up \times \text{rs}g \times down))) \\
\text{until} & \text{ fixpoint reached}
\end{align*}
\]

\[
rs^{i+1} := rs^i \cup flat \cup \pi_{16}(\sigma_{2=4}(\sigma_{3=5}(up \times rs \times down)))
\]

Level 0: \( \emptyset \)
Level 1: \( \{(g, f), (m, n), (m, o), (p, m)\} \)
Level 2: \( \{\text{Level 1}\} \cup \{(a, b), (h, f), (i, f), (j, f), (j, k)\} \)
Level 3: \( \{\text{Level 2}\} \cup \{(a, c), (a, d)\} \)
Level 4: \( \{\text{Level 3}\} \)
Naïve Algorithm for Evaluating Datalog Programs

- redundant computations (all elements of the preceding level are taken into account)
- on each level, all elements of the preceding level are re-computed
- monotone (rsg is extended more and more)
Agenda

• Rules
  – Llyod-Topor Transformation

• Datalog
  – Characterizations of Datalog Program Semantics

• Evaluating Datalog Programs
  – Naïve Evaluation
  – Semi-naïve Evaluation

• Rules for RDFS via a Triple Predicate

• Rules for RDFS via Direct Translation
Semi-Naïve Algorithm for Computing \( rsg \)

focus on facts that have been newly computed on the preceding level

**Algorithm 2** \( RSG' \)

\[
\begin{align*}
\Delta_{rsg}^1(x, y) & := flat(x, y) \\
\Delta_{rsg}^{i+1}(x, y) & := up(x, x_1), \Delta_{rsg}^i(y_1, x_1), down(y_1, y)
\end{align*}
\]

- not recursive
- no Datalog program (set of rules is infinite)
- for each input \( I \) and \( \Delta_{rsg}^i \) (the newly computed instances on level \( i \)),

\[
\Delta_{rsg}^{i+1} \subseteq \Delta_{rsg}^i \subseteq rsg^{i+1}
\]

- \( RSG(I)(rsg) = \bigcup_{1 \leq i} (\Delta_{rsg}^i) \)
- less redundancy
An Improvement

But: $\Delta_r^{i+1} \neq rsg^{i+1} - rsg^i$

e.g.: $(g,f) \in \Delta_r^2, (g,f) \notin rsg^2 - rsg^1$
$\leadsto rsg(g,f) \in rsg^1$, because $flat(g,f)$,
$\leadsto rsg(g,f) \in \Delta_r^2$, because $up(g,n), rsg(m,f), down(m,f)$

- idea: use $rsg^i - rsg^{i-1}$ instead of $\Delta_r^i$ in the second “rule” of $RSG'$

**Algorithm 3 $RSG''$**

\[
\begin{align*}
\Delta_1^{rsg}(x,y) &:= flat(x,y) \\
rsg^1 &:= \Delta_1^{rsg} \\
tmp^{i+1}_{rsg}(x,y) &:= up(x,x_1), \Delta^{i}_{rsg}(y_1,x_1), down(y_1,y) \\
\Delta^{i+1}_{rsg}(x,y) &:= tmp^{i+1}_{rsg} - rsg^i \\
rsg^{i+1} &:= rsg^i \cup \Delta^{i+1}_{rsg}
\end{align*}
\]
Agenda

• Rules
  – Llyod-Topor Transformation
• Datalog
  – Characterizations of Datalog Program Semantics
• Evaluating Datalog Programs
  – Naïve Evaluation
  – Semi-naïve Evaluation
• Rules for RDFS via a Triple Predicate
• Rules for RDFS via Direct Translation
Datalog Rules for RDFS (no Datatypes & Literals)

problem: no strict separation between data and schema (predicates)

\[ a \ rdfs:domain \ x . \ u \ a \ y . \]
\[ u \ rdf:type \ x . \ rdfs2 \]

\[ rdf:type(u, x) \leftarrow rdfs:domain(a, x) \land a(u, y) \]

- solution: use a triple predicate
Datalog Rules for RDFS (no Datatypes & Literals)

problem: no strict separation between data and schema (predicates)

\[
\begin{align*}
\text{a } \text{rdfs:domain } x . & \quad \text{u } a \quad y . \\
\text{u } \text{rdf:type } x . & \quad \text{rdfs2}
\end{align*}
\]

\[
\text{rdf: type}(u, x) \leftarrow \text{rdfs: domain}(a, x) \land a(u, y)
\]

- solution: use a triple predicate
Agenda

- Rules
  - Lloyd-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation
Datalog Rules for RDFS (no Datatypes & Literals)

\[
\begin{align*}
\text{a rdfs:domain } x . & \ u a y . \\
\text{    u rdf:type } x . & \quad \text{rdfs2}
\end{align*}
\]

\( Triple(u, \text{rdf: type, } x) \leftarrow Triple(a, \text{rdfs: domain, } x) \land Triple(u, a, y) \)

- usage of just one predicate reduces optimization potential
- all (newly derived) triples are potential candidates for any rule
- rules change when the data changes, no separation between schema and data
Datalog Rules for RDFS (no Datatypes & Literals)

- solution 2: introduce specific predicates

\[
\begin{align*}
\text{a rdfs:domain} & \ x \ . \ u \ a \ y \ . \\
\text{u rdf:type} & \ x \ . \ rdfs2
\end{align*}
\]

\[
\text{type}(u, x) \leftarrow \text{domain}(a, x) \land \text{rel}(u, a, y)
\]
Axiomatic Triples as Facts

type(rdf:type, rdf:Property)
type(rdf:subject, rdf:Property)
type(rdf:predicate, rdf:Property)
type(rdf:object, rdf:Property)
type(rdf:first, rdf:Property)
type(rdf:rest, rdf:Property)
type(rdf:value, rdf:Property)
type(rdf:_1, rdf:Property)
type(rdf:_2, rdf:Property)
type(..., rdf:Property)
type(rdf:nil, rdf:List)
... (plus RDFS axiomatic triples)
Axiomatic Triples as Facts

\[
type(rdf:type, rdf:Property) \\
type(rdf:subject, rdf:Property) \\
type(rdf:predicate, rdf:Property) \\
type(rdf:object, rdf:Property) \\
type(rdf:first, rdf:Property) \\
type(rdf:rest, rdf:Property) \\
type(rdf:value, rdf:Property) \\
type(rdf:_1, rdf:Property) \\
type(rdf:_2, rdf:Property) \\
type(..., rdf:Property) \\
type(rdf:nil, rdf:List) \\
\ldots \text{ (plus RDFS axiomatic triples)}
\]
Axiomatic Triples as Facts

type(rdf:type, rdf:Property)
type(rdf:subject, rdf:Property)
type(rdf:predicate, rdf:Property)
type(rdf:object, rdf:Property)
type(rdf:first, rdf:Property)
type(rdf:rest, rdf:Property)
type(rdf:value, rdf:Property)
type(rdf:_1, rdf:Property)
type(rdf:_2, rdf:Property)
type(..., rdf:Property)
type(rdf:nil, rdf:List)

...(plus RDFS axiomatic triples)

⇝ only needed for those rdf:_i that occur in the graphs \( G_1 \) and \( G_2 \), if \( G_1 \models ? \ G_2 \) is to be decided
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
    u & a & y \\
    \text{a rdf:type rdf:Property} \quad \text{rdf1} \\
\Rightarrow & \text{type(a, rdf:Property) } \leftarrow \text{rel(u, a, y)}
\end{align*}
\]

\[
\begin{align*}
    a, b & \text{ IRIs} \\
    x, y & \text{ IRI, blank node or literal} \\
    u, v & \text{ IRI or blank node} \\
    \_ & \text{ literal} \\
    \_:n & \text{ blank nodes}
\end{align*}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[ \begin{align*} 
\text{rdf1:} & \quad \text{type}(a, \text{rdf:Property}) \leftarrow \text{rel}(u, a, y) \\
\text{rdfs2:} & \quad \text{domain}(a, x) \land \text{rel}(u, a, y) \quad \text{type}(u, x) \leftarrow \text{domain}(a, x) \land \text{rel}(u, a, y)
\end{align*} \]

\( a, b \) IRIs \quad \( x, y \) IRI, blank node or literal
\( u, v \) IRI or blank node \quad \text{literal} \quad \_n \) blank nodes

TU Dresden \quad Foundations of Semantic Web Technologies
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
  & \frac{u \ a \ y}{\text{a rdf:type rdf:Property}} \quad \text{rdf1} \\
  & \leadsto \text{type(a, rdf:Property)} \leftarrow \text{rel(u, a, y)} \\
  & \frac{\text{a rdfs:domain x} \ . \ u \ a \ y}{\text{u rdf:type x}} \quad \text{rdfs2} \\
  & \leadsto \text{type(u, x)} \leftarrow \text{domain(a, x)} \land \text{rel(u, a, y)} \\
  & \frac{\text{a rdfs:range x} \ . \ u \ a \ v}{\text{v rdf:type x}} \quad \text{rdfs3} \\
  & \leadsto \text{type(v, x)} \leftarrow \text{range(a, x)} \land \text{rel(u, a, v)}
\end{align*}
\]

a, b IRIs  
\text{x, y IRI, blank node or literal}  
u, v IRI or blank node  
l literal  
\_\_\_n blank nodes
RDF Entailment Rules (no Datatypes & Literals)

\[
\text{\underline{rdf1}} \\
\frac{u a y}{a \text{ rdf:type rdf:Property}} \quad \text{type}(a, \text{rdf:Property}) \leftarrow \text{rel}(u, a, y)
\]

\[
\text{\underline{rdfs2}} \\
\frac{a \text{ rdfs:domain } x . u a y .}{u \text{ rdf:type } x .} \\
\text{\underline{rdfs3}} \\
\frac{a \text{ rdfs:range } x . u a v .}{v \text{ rdf:type } x .} \\
\text{\underline{rdfs4a}} \\
\frac{u a x .}{u \text{ rdf:type rdfs:Resource} .} \\
\Rightarrow \text{type}(u, \text{rdfs:Resource}) \leftarrow \text{rel}(u, a, x)
\]

\[
a, b \text{ IRIs} \\
x, y \text{ IRI, blank node or literal} \\
u, v \text{ IRI or blank node} \\
\_n \text{ blank nodes}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[
\frac{u \ a \ v}{\top} \quad \text{rdfs4b}
\]

\[v \text{ rdf:type } \text{rdfs:Resource} \quad \text{rdfs4b}
\]

\[
\sim \text{ type}(v, \text{rdfs:Resource}) \leftarrow \text{rel}(u, a, v)
\]

\[
a, \ b \text{ IRIs} \quad x, \ y \text{ IRI, blank node or literal}
\]

\[
u, \ v \text{ IRI or blank node} \quad \text{l literal} \quad \ldots \text{n blank nodes}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{u} & \text{ a v .} \\
\text{v} & \text{ rdf:type rdfs:Resource} . \\
\text{\implies \ type}\text{(v, rdfs:Resource)} & \leftarrow \text{rel}\text{(u, a, v)}
\end{align*}
\]

\[
\begin{align*}
\text{u} & \text{ rdfs:subPropertyOf v .} \\
\text{v} & \text{ rdfs:subPropertyOf x .} \\
\text{\implies \ subPropertyOf}\text{(u, x)} & \leftarrow \text{subPropertyOf}\text{(u, v)} \land \text{subPropertyOf}\text{(v, x)}
\end{align*}
\]

\[a, b \text{ IRI, } x, y \text{ IRI, blank node or literal} \]
\[u, v \text{ IRI or blank node } l \text{ literal } \\n\text{_:n blank nodes} \]
RDF Entailment Rules (no Datatypes & Literals)

\[ \begin{align*}
  u & \equiv v . \\
  v & \text{ rdf:type rdfs:Resource} . \\
  \text{type}(v, \text{rdfs:Resource}) & \leftarrow \text{rel}(u, a, v) \\

  u & \text{ rdfs:subPropertyOf } v . \\
  v & \text{ rdfs:subPropertyOf } x . \\
  \text{subPropertyOf}(u, x) & \leftarrow \text{subPropertyOf}(u, v) \land \text{subPropertyOf}(v, x) \\

  u & \text{ rdf:type rdf:Property} . \\
  u & \text{ rdfs:subPropertyOf } u . \\
  \text{subPropertyOf}(u, u) & \leftarrow \text{type}(u, \text{rdf:Property})
\end{align*} \]

\( a, b \) IRIs \hspace{1cm} x, y \ IRI, blank node or literal
\( u, v \) IRI or blank node \hspace{1cm} \text{literal} \hspace{1cm} \_n \) blank nodes
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{rdfs4b} & \quad \frac{u \ a \ v}{v \ \text{rdf:type} \ \text{rdfs:Resource}} \quad \rightarrow \quad \text{type}(v, \text{rdfs:Resource}) \leftarrow \text{rel}(u, a, v) \\
\text{rdfs5} & \quad \frac{u \ \text{rdfs:subPropertyOf} \ v \quad \text{v \ rdfs:subPropertyOf} \ x}{u \ \text{rdfs:subPropertyOf} \ x} \quad \rightarrow \quad \text{subPropertyOf}(u, x) \leftarrow \text{subPropertyOf}(u, v) \land \text{subPropertyOf}(v, x) \\
\text{rdfs6} & \quad \frac{u \ \text{rdf:type} \ \text{rdf:Property}}{u \ \text{rdfs:subPropertyOf} \ u} \quad \rightarrow \quad \text{subPropertyOf}(u, u) \leftarrow \text{type}(u, \text{rdf:Property}) \\
\text{rdfs7} & \quad \frac{a \ \text{rdfs:subPropertyOf} \ b \quad u \ a \ y}{u \ b \ y} \quad \rightarrow \quad \text{rel}(u, b, y) \leftarrow \text{subPropertyOf}(a, b) \land \text{rel}(u, a, y)
\end{align*}
\]

TU Dresden Foundations of Semantic Web Technologies
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
& u \text{ rdf:type } \text{rdfs:Class .} \\
& u \text{ rdf:subClassOf } \text{rdfs:Resource .} \\
& \sim \text{subClassOf}(u, \text{rdfs:Resource}) \iff \text{type}(u, \text{rdfs:Class})
\end{align*}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{rdfs8} & \quad \text{u rdf:type rdfs:Class} \land \text{v rdf:type u} \\
\text{rdfs9} & \quad \text{v rdf:type x} \\
\Rightarrow & \quad \text{subClassOf(u, rdfs:Resource) } \leftarrow \text{type(u, rdfs:Class)} \\
\Rightarrow & \quad \text{subClassOf(u, x) } \land \text{type(v, x)} \leftarrow \text{type(v, x)}
\end{align*}
\]

a, b IRIs  
\( x, y \) IRI, blank node or literal  
u, v IRI or blank node  
l literal  
\( _n \) blank nodes

TU Dresden  
Foundations of Semantic Web Technologies
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{u rdf:type rdfs:Class} & . \\
\text{u rdf:subClassOf rdfs:Resource} & . \\
\rightarrow \text{subClassOf(u, rdfs:Resource) } \leftarrow \text{type(u, rdfs:Class)}
\end{align*}
\]

\[
\begin{align*}
\text{u rdfs:subClassOf x . v rdf:type u} & . \\
\text{v rdf:type x} & . \\
\rightarrow \text{type(v, x) } \leftarrow \text{subClassOf(u, x) } \land \text{type(v, x)}
\end{align*}
\]

\[
\begin{align*}
\text{u rdf:type rdfs:Class} & . \\
\text{u rdfs:subClassOf u} & . \\
\rightarrow \text{subClassOf(u, u) } \leftarrow \text{type(u, rdfs:Class)}
\end{align*}
\]

\[
\begin{align*}
a, b \text{ IRI}s & . \\
x, y \text{ IRI, blank node or literal} & . \\
u, v \text{ IRI or blank node} & . \\
literal & . \\
\_n \text{ blank nodes} & .
\end{align*}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[ u \text{ rdf:type rdfs:Class} . \quad \text{rdfs8} \]
\[ u \text{ rdf:subClassOf rdfs:Resource} . \quad \text{rdfs8} \]
\[ \text{⇝ subClassOf}(u, \text{rdfs:Resource}) \leftarrow \text{type}(u, \text{rdfs:Class}) \]

\[ u \text{ rdfs:subClassOf} x . \quad v \text{ rdf:type} u . \quad \text{rdfs9} \]
\[ v \text{ rdf:type} x . \]
\[ \text{⇝ type}(v, x) \leftarrow \text{subClassOf}(u, x) \land \text{type}(v, x) \]

\[ u \text{ rdf:type rdfs:Class} . \quad \text{rdfs10} \]
\[ u \text{ rdfs:subClassOf} u . \]
\[ \text{⇝ subClassOf}(u, u) \leftarrow \text{type}(u, \text{rdfs:Class}) \]

\[ u \text{ rdfs:subClassOf} v . \quad v \text{ rdfs:subClassOf} x . \quad \text{rdfs11} \]
\[ u \text{ rdfs:subClassOf} x . \]
\[ \text{⇝ subClassOf}(u, x) \leftarrow \text{subClassOf}(u, v) \land \text{subClassOf}(v, x) \]

\[ a, b \text{ IRI}s \quad x, y \text{ IRI, blank node or literal} \]
\[ u, v \text{ IRI or blank node} \quad \text{literal} \quad n \text{ blank nodes} \]

TU Dresden Foundations of Semantic Web Technologies
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{u} & \text{ rdf:} \text{type} \text{ rdfs:} \text{ContainerMembershipProperty}. \text{ rdfs12} \\
\text{u} & \text{ rdfs:} \text{subPropertyOf} \text{ rdfs:} \text{member}. \\
\text{\sim} & \text{ subPropertyOf}(\text{u}, \text{rdfs:} \text{member}) \leftarrow \text{type}(\text{u}, \text{rdfs:} \text{ContainerMembershipProperty})
\end{align*}
\]

\[\begin{align*}
a, b & \text{ IRIs} \quad x, y \text{ IRI, blank node or literal} \\
u, v & \text{ IRI or blank node} \quad \text{literal} \quad :n \text{ blank nodes}
\end{align*}\]
Agenda

• Rules
  – Lloyd-Topor Transformation

• Datalog
  – Characterizations of Datalog Program Semantics

• Evaluating Datalog Programs
  – Naïve Evaluation
  – Semi-naïve Evaluation

• Rules for RDFS via a Triple Predicate

• Rules for RDFS via Direct Translation